Migrating Workstation Applications between Windows NT* and UNIX* Systems

Comparing Methodologies, Features and Interoperability

Version 1.0 Initial Release © Intel Corporation 1999

Table of Contents

Abstract

Much of the technical workstation software to be ported onto Microsoft* Windows NT* originates on UNIX* systems, consists of a very large number of lines of C (or some C dialect) employing C function and UNIX system calls, and performs on-screen graphics with X11. While the UNIX and Win32 OS environments can interoperate (via TCP/IP networking, DNS, file sharing, etc.), the UNIX and Win32 programming environments (i.e., the APIs) are quite different. The cost of converting any UNIX-oriented source to Win32 can be high—education, partitioning, re-coding, debugging, re-testing, etc. A number of different strategies are available to ease such ports, including UNIX emulation environments, application partitioning, and standardization and X11 servers. Many such scenarios are described and compared in this paper, including a comparison of DataFocus* NuTcracker* and Softway Systems* Interix*.

The Problem

Migration of professional workstation applications from UNIX to Windows NT is complicated by a number of factors, which are listed and explored in the following sections.

Dependence on UNIX* Infrastructure

UNIX has long been the base on which professional workstation applications have been written in a wide variety of industries: MCAD[1], EDA[2], DCC[3], financial, scientific computation, etc. The demands of these applications are significant in terms of problem size, code size, data processing speed, and iterative algorithms. They require workstation capabilities historically met only by UNIX-based workstations from vendors like Sun Microsystems, Hewlett-Packard, Silicon Graphics, Digital, IBM and others.

The applications that have been written for the UNIX workstation environment:

- Most often use C or FORTRAN (or a C dialect like C++)
- Make C library and UNIX system calls
- May depend on the UNIX process or signaling model
- Use TCP/IP for networking

[1] Mechanical Computer Aided Design (MCAD)

[2] Electronic Design Automation (EDA), also called Electronic Computer Aided Design (ECAD)

[3] Digital Content Creation (DCC), the creation or editing of digital imagery and animation

- Employ X11 (or sometimes dedicated, proprietary) graphics
- Employ shared libraries and dynamic loading

Single Source Benefits

Because the UNIX market segment is fragmented, independent software vendors (ISVs) must usually offer their applications on several platforms and UNIX variants to have adequate market segment size. Because of UNIX standardization efforts, the differences between UNIX implementations can usually be contained within conditional compilation, all driven from one source code and build rule repository, and shared among all the UNIX target environments. (This is usually referred to as an application being "*single source*.") This is particularly important because it is not uncommon for source code to be measured in millions of lines; the support burden of maintaining separate source code bases is unmanageable for most ISVs.

Legacy Installations and Customer Data

Because the professional workstation application marketplace has been in operation for many years, any customer site is a mixture of old and new hardware, often from several vendors and usually expensive. Projects being done on these workstations are long-lived and typically build upon previous projects' data, tools or machines. Thus, new workstation, OS or application offerings must co-exist with this legacy UNIX environment—sharing data, source code, networking, servers and the like.

Complicated or Customized Build Mechanism

UNIX has a very flexible software build environment that includes multi-platform and parallel capabilities. Professional workstation application providers take *full* advantage of all these capabilities.

Decision-Making Influences in Migration Planning

The market segment penetration, competitive speed, price advantage and functional completeness of Windows*/NT on Intel® platforms now make migration to that environment imperative. There are many ways to get there, however, each influenced by product marketing goals and existing application internal and build architectures.

Migration Goals

- Preserve existing intellectual capital (algorithms, APIs, data structures, etc.)
- Make migration timely
- Minimize increase in complexity (of application or build mechanisms)

- Preserve existing support mechanisms
- Educate technical staff in new operating paradigms at a comfortable, supportable rate
- Remain consistent over time with product-line primary target platform plans

Market Segment Issues

An appropriate migration strategy can be chosen *only* with adequate market segment research. This includes a thorough knowledge of the relationship of the potentially-migrated application to others in a *family* of tools—potentially from a variety of providers—used by the end user to accomplish an overall task. Ask *at least* the following questions:

- What is the expected lifetime of the migrated application?
    - How much money is appropriate to spend in doing the migration?
    - Can migration occur over several releases of the product?
- What is the urgency of getting the product onto Windows NT?
- Will the entire product line be migrated to Windows NT over time?
    - If so, Windows-invasive porting is more appropriate.
    - If not, product modularization can constrain amount of migration work.
- Should the ported application be Windows logo-compliant?
    - Would the legacy GUI be a market segment impediment—or an asset?
    - Will day-to-day users experience both legacy and Windows environments?
    - Is a Windows GUI an eventual target for the application? If so, how soon?
    - Will it make sense to have updated versions of the UNIX product have a Windows look-and-feel GUI as well?
- Will personnel have to be extensively retrained to support a migrated product?
- To what degree will the UNIX and Windows products interoperate: data only, client/server, etc.?
    - Will the Windows NT desktop be used to control non-migrated and/or remotely running applications?
    - What other applications will be used in conjunction with the migrated one?
    - Are companion products available in both UNIX and Windows forms?
- What are the costs and risks of using third-party porting aids?

  - Are the costs of porting aids—whether in development-time or for seat licensing—acceptable in comparison to the product's ASP[4]?
  - Are interfaces to porting aids proprietary or are they public, multi-source APIs? If proprietary, is there adequate safety in using them and becoming dependent upon them?
  - Are UNIX-to-NT aids more expensive than NT-to-UNIX aids? Consider both initially and over time, taking product lifetime into consideration.
- Are third-party software providers key to product delivery?
  - Does the product have "add-ins" or modules coming from third parties? What will they feel comfortable doing about their code?
  - Is the product used with extensive scripts and/or macros? What is the cost of revising or migrating them?

Application Architectures

Application architecture—or architectural revisions—can dramatically affect the ease of migration. (Often a companion benefit of architectural revisions is improved product maintainability and market segment flexibility—so it is good to do in any event.) There are several basic styles of application construction, discussed further below. The following questions can help determine application migration strategies:

- How extensively is UNIX used (vs. just-plain ANSI C library calls)?
  - I/O: ANSI C's `<stdio.h>` vs. UNIX' `open()`, `read()`, etc.?
  - Asynchronous operation: Signals? POSIX threads?
  - Process semantics: `fork()`, `exec()` vs. `spawn()`—i.e., tightly paired `fork()` and `exec()`?
- Is the application already client/server partitioned?
  - What IPC mechanism is used: Berkeley sockets, RPC, etc.?
- Can the application GUI be easily separated from the application body?
  - How does the GUI communicate with the application body?
  - How is the existing product internationalized?
- How is application data stored?

[4] Average Sales Price

  - If text, how are lines terminated?
  - If binary, is the data little- or big-endian[5]? Naturally aligned? Compressed? Encoded (as XDR)?
- How does the application use dynamic linking?
  - Do third parties supply significant portions of the application technology?
  - Are dynamic loadables used by multiple applications?
- Do third-parties supply parts of the application?
  - Are application extensions runtime-loaded or statically linked?
  - Are other applications sub-invoked to do any tasks?

The Monolithic Application

This kind of application is common in EDA and scientific computing, where data is batch-read, operated upon and then batch-written. The so-called "GUI" can sometimes be as simple as just a command-line interface.

A variety of ways to migrate these monolithic applications are discussed below, including several UNIX emulation environments.

Client/Server Application

This can be viewed as a monolithic application that has been split into two parts, intercommunicating (typically with Berkeley sockets and TCP/IP) to distribute or parallelize the workload.

Client/server applications offer the opportunity (when used in a heterogeneous environment of UNIX and Windows NT machines) of migrating the client (desktop) and server portions of the software on different schedules. If communication between the partitions has *not* been done with basic/bare Berkeley sockets (i.e., it uses Sun or DCE RPC, or some third-party facility), the migration task is larger.

[5] Big-endian is Motorola byte order—i.e., most significant byte of data stored in the byte with the lowest address; little-endian is Dec/Intel byte order— least significant byte of data stored in the byte with the lowest address. A big-endian machine cannot straightforwardly read data written by a little-endian machine (and vice versa); application software must do format recognition and conversion.

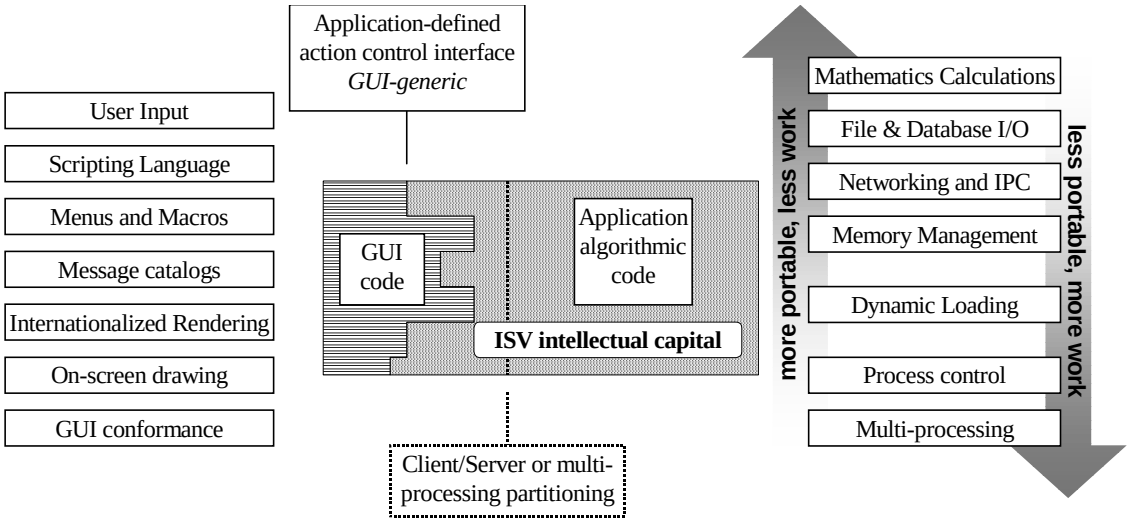


Figure 1: Generalized Monolithic Application

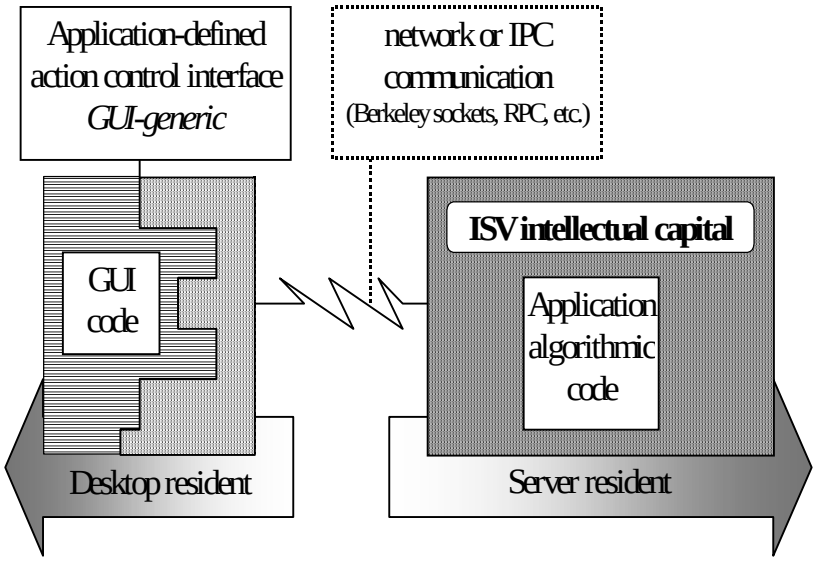


Figure 2: Client/Server Application

The client/server application can be treated as a special case of the monolithic application by applying migration decision criteria to each part (client or server) independently.

Framework/Add-in Application

Professional workstation applications are often collaborations of software from a variety of ISVs, which, together, perform the customer task. In EDA, this may mean a simulator is composed of several "engines" working at different abstraction levels (analog, gate, behavioral), simulation runtime models from several third parties, and test vector handling software from yet another.

Such applications are often organized as a basic framework application that can extend itself with OEM'd code libraries or at runtime according to scripts or application data.

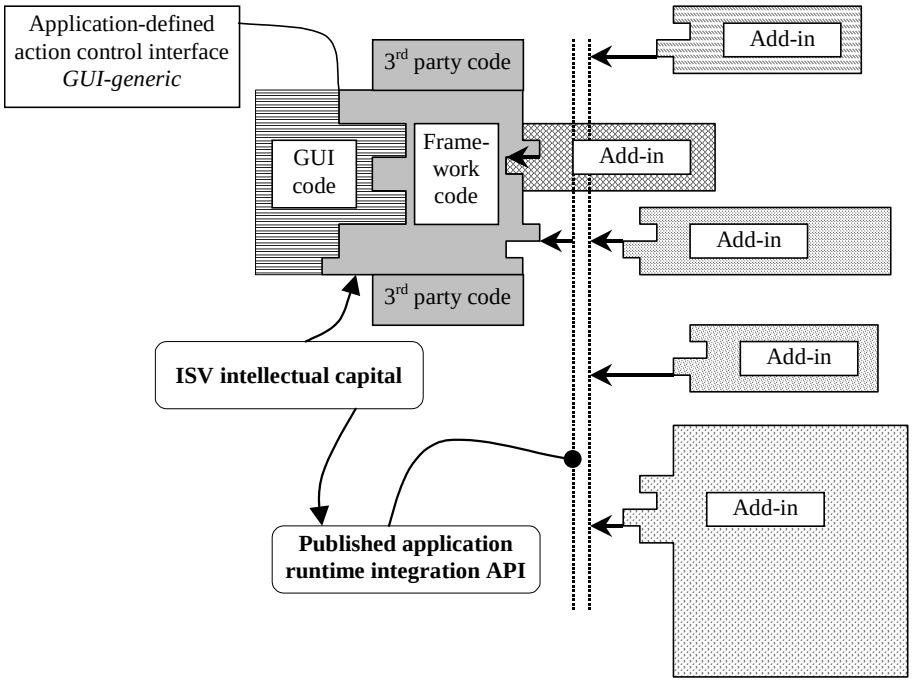


Figure 3: Application as a Framework

This situation can be difficult; the third-party suppliers must have their add-ins available on Windows NT and they must usually be available via the same migration strategy as the framework application. Once agreement is negotiated (usually a lowest-common-denominator set of features), the approaches described for the monolithic application can be applied to all parts uniformly. Use of remote procedure call facilities like DCE or COM can ease this situation, but usually at the cost of efficiency.

GUI Migration Strategies

GUIs have standards for appearance, user gestures, event handling and notification and system interaction that make an application a good citizen on their host platform. The user then sees a comfortable similarity in operation of applications across the whole system, and can use OS-intermediated communication between running software (cut/paste, input editing, real-time data exchange, etc.).

The primary decision regarding GUI migration is the degree of host GUI compliance. This leads to 4 possible migration strategies:

1. Compliant GUIs for each environment
2. Standard Motif/X11 GUI everywhere
3. Standard Windows logo-compliant GUI everywhere
4. Portability via an independent GUI abstraction

These are illustrated below, then discussed further. An additional case:

5. Use of a proprietary GUI framework

is not discussed; this is becoming uncommon due to development, support and training costs.

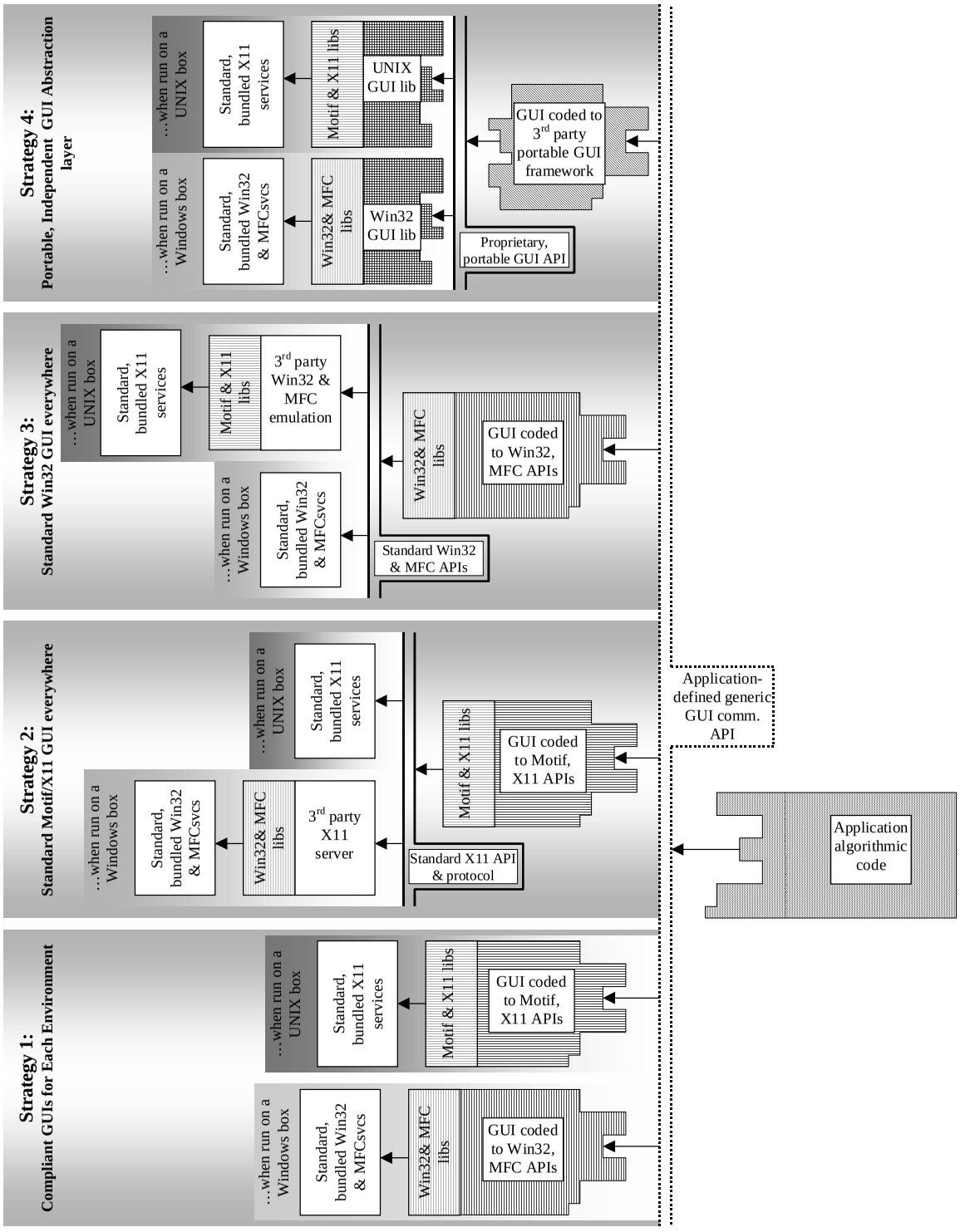


Figure 4: GUI Migration Strategies

GUI Porting Strategy: (Criteria, issue or topic)	**Compliant GUI for each environment**	**Standard Motif/X11 GUI everywhere**	**Standard Windows logo-compliant GUI everywhere**	**Portability via an independent GUI abstraction**
Example	Native UNIX and Native Windows	Native UNIX *plus* SCO Xvision or other X server, possible UNIX emulation env.	Native Windows *plus* Bristol Wind/U, MainSoft MainWin, Willows, etc.	Tcl/Tk, TeleUSE, Allegris[6], Java
Value	Standard behavior and implementation on each platform, best native platform interoperability (cut/paste, etc.)	Standard behavior and implementations familiar to UNIX users; write-it-once GUI code	Standard behavior and implementations familiar to Windows users; write-it-once GUI code	Write-it-once GUI code
Risk	Minimized transfer of usage skills between UNIX and Windows users; added cost of two GUI teams	Emulation of Motif/X11 on Windows may not be "perfect" (event ordering, mouse gestures, etc.) or may perform poorly	Emulation of Win32 on UNIX may not be "perfect" (event ordering, mouse gestures, etc.) or may perform poorly; emulation supplier "health"	*Non*-standard; Continued viability of third-party supplier; adopting support burden of "freeware"; lowest-common-denominator capabilities; delayed feature arrival

[6] It appears that Allegris (formerly C++/Views) is no longer a supported product.

GUI Porting Strategy: (Criteria, issue or topic)	**Compliant GUI for each environment**	**Standard Motif/X11 GUI everywhere**	**Standard Windows logo-compliant GUI everywhere**	**Portability via an independent GUI abstraction**
Future trends	Separate development tracks for each GUI, product lifetime is just a market segment decision	Gradually decreasing attractiveness over time as Windows boxes (therefore that GUI) proliferates on the desktop; suppliers are stable and reliable	Increasing attractiveness as Windows desktops proliferate	Decreasing availability of these kinds of products
Dependencies	Nothing more than platform vendor's native offerings (good)	Third-party supplier of Motif/X11 only on Windows	Third-party supplier of Win32 on only UNIX	Third-party supplier for code, libraries and support on *both* platform types
Efficiency	Maximal on each platform	Lower on Windows	Lower on Motif/X11	Acceptable on either platform
Functionality	Best functionality in each environment	Functionality may be "different" or slightly subsetted on Windows platform	Functionality may be "different" or slightly subsetted on UNIX platforms	Often lowest-common-denominator (only widgets, macros, etc. implementable in both worlds)
Portability	Minimal issue	Have to obey some substetting constraints; good platform availability	Have to obey some substetting constraints; limited platform availability (no SunOS 4, …)	Constrained to platforms on which GUI abstraction is offered

GUI Porting Strategy: (Criteria, issue or topic)	**Compliant GUI for each environment**	**Standard Motif/X11 GUI everywhere**	**Standard Windows logo-compliant GUI everywhere**	**Portability via an independent GUI abstraction**
Supportability	Minimal on each platform, but requires two support tracks; minimal transferal of user knowledge	Standard risks of working with a third-party and an emulated environment	Standard risks of working with a third-party and an emulated environment	Dependence on single-source supplier
Legacy Preservation	Any existing GUI mostly preserved	Any existing Motif GUI mostly preserved	Any existing Windows GUI mostly preserved	If already using one of these, almost all is preserved (until time to standardize)
New work	Write one Windows GUI from scratch	If already doing Motif/X11, only integration and test; if *not*, write Motif/X11 GUI from scratch	If already doing Windows GUI, only integration and test; if *not*, write Windows GUI from scratch	If already using one of these, only integration and test; if *not* already using one, write (one) new GUI from scratch
Organizational	Two separate GUI groups doing GUIs for all apps: Motif experts and Win32 experts; retain existing Motif coders and get Windows ones	Need specialists for the Motif/X11 emulation on Windows in engineering and support groups; Motif coders existent or can be hired	Need specialists for the Win32 emulation on UNIX in engineering and support groups; Windows coders easy to hire	Need locally-trained specialists on third-party GUI abstraction; essentially no ability to hire already-trained personnel

Application Core Migration Strategies

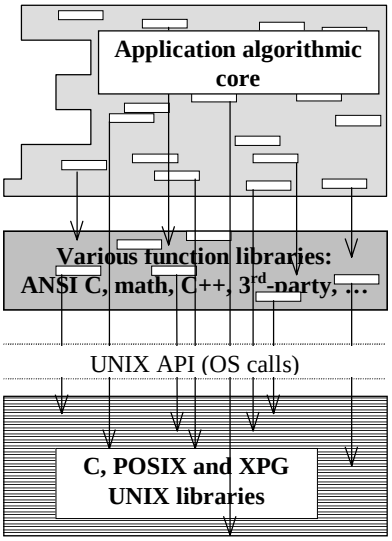


Figure 5: Random Calls to Services

For most applications to be migrated from UNIX to Windows/NT, the most OS-specific code occurs in the GUI (discussed above). By contrast, the algorithmic core consists largely of code that uses function libraries that are *much less* OS-specific: the ANSI C library, various math libraries, standard C++ libraries, etc.

Usually, one finds calls to these functions scattered randomly throughout application source, as in the illustration. Even so, there are some functions that are more portable than others: for example the highly portable `memcpy()` vs. the problematic `fork()`, which has no direct Windows NT equivalent.

It is usually worthwhile to spend resources to do some code re-work which allows the code to be more easily migrated to Windows/NT—or, at least, to be more portably organized, because:

- The ISV's intellectual property is concentrated in this application algorithmic core
- It is more portable that most GUI code
- There are actually very few calls to the less-portable functions

Note: This usually assists ports among UNIX variants also.

Reorganization of Source for Easier Migration

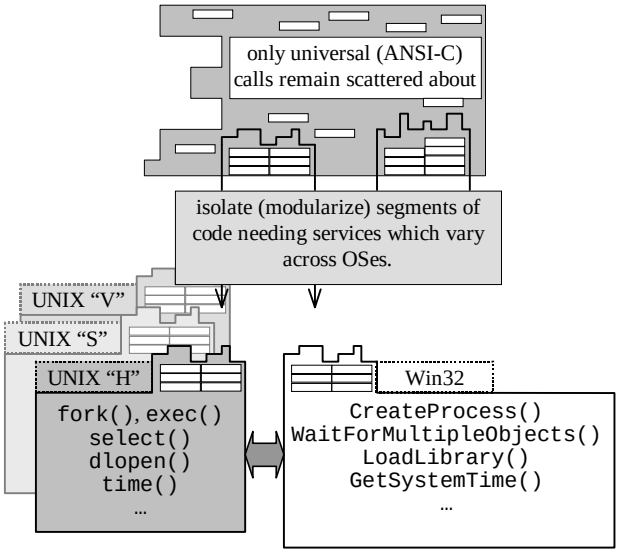


Figure 6: Modularize Non-Portabilities

The modules of application source can be reorganized to sequester less portable calls and algorithms into specialized modules that all have the same call interface. The different modules can be implemented in a variety of ways:

- Conditional compilation, when variances are modest, as with UNIX-to-UNIX differences
- Separate compilation, when internal variances are larger, as between UNIX and Windows. In most cases, the largest body of source remains the portable part, having calls to simple ANSI C, math, standard C++ or other OS-independent libraries in it.

Note that the logical equivalent of this is *already* necessary for applications available on multiple UNIX systems. Today's UNIX-to-UNIX variances must be already accounted for, thus much of this work may already be done.

Migration via UNIX Emulation: Minimizing Work

As Windows/NT is a functionally complete, modern operating system, it contains analogues to nearly every UNIX capability[7]. Thus, it is possible to write a translation layer that emulates the UNIX API. Several of these are available[8], including:

- DataFocus' NuTcracker
- Softway Systems' Interix
- the GNU/Cygnus Win32GNU environment

[7] Several of these would not have been implemented were it not for Windows/NT's POSIX subsystem. Its presence caused just enough "extra" features to be added to the Windows/NT kernel to make complete UNIX emulation possible; for example, deletability of files within a writeable directory and support in the kernel for address space replication (for fork()).

[8] Consensys formerly offered a product named Portage, but it appears to no longer be available. AT&T has a package named U/Win, but it is not (yet?) commercially available.

(Note: we're not defining a new, unique API nor using some company's proprietary one; we're using a standard-defined one: the UNIX API. They are further discussed and compared in the Table below.)

Topic, Feature or Characteristic:	**NuTcracker**	**Interix**	**Win32GNU**
Supplier	DataFocus Inc. Suite 400 12450 Fair Lakes CirFairfax, VA 22033-3813 800-637-8034 http://www.datafocus.com	Softway Systems, Inc. Suite 5514 185 Berry Street San Francisco, CA 94107 800-GET-UNIX http://www.interix.com	Cygnus Solutions 1325 Chesapeake Terrace Sunnyvale, CA 94089 408-542-9600 http://www.cygnus.com/misc/gnu-win32/
Description	Nearly UNIX-brandable API and command framework operating within the (native) Win32 subsystem, allowing concurrent access to UNIX & Win32 features; includes X server (and optionally X/Motif SDK)	Nearly UNIX-brandable replacement for the Windows/NT POSIX subsystem - commands, APIs, X server, compiler, debugger, etc. - which does *not* currently allow direct access to the Win32 API - but access to *any* NT process	(Loosely) standards-targeted UNIX API and command framework operating within the (native) Win32 subsystem, allowing concurrent access to UNIX & Win32 features; no X server
Status	Release 4.1 now available	Release 2.2 now available	Beta (only) ß19 now available
Approximate Cost	O($5K), plus royalties	O($500)	freeware[9]
Outsourced sub-parts	Commands OEM'd from MKS SCO Xvision X server	Choice of X servers; NFS clients	Other freeware ports of UNIX tools; X server must be independently obtained

[9] "Freeware" is software whose source code is public available and must remain so, and is often very highly functional. However, it is *not* supported like other commercial software: updates come from other users via the net, your dedicated employees or via contracted support from firms like Cygnus Solutions.

Topic, Feature or Characteristic:	**NuTcracker**	**Interix**	**Win32GNU**
Content (compared to UNIX/98 specification)	UNIX/98 commands, UNIX/95 APIs, threads now available, some missing APIs:	UNIX/98 commands, UNIX/95 APIs, threads soon to be available, some missing APIs:	GNU commands, older UNIX APIs migrating toward UNIX/95, many APIs missing

NuTcracker Architecture (and also Win32GNU)

The figure below shows a minimally changed application employing the NuTcracker framework. Note that the application can make Win32 API calls, allowing it to adopt, inject or conditionally use those calls as convenient. While individual details differ, the overall architecture of Win32GNU is about the same.

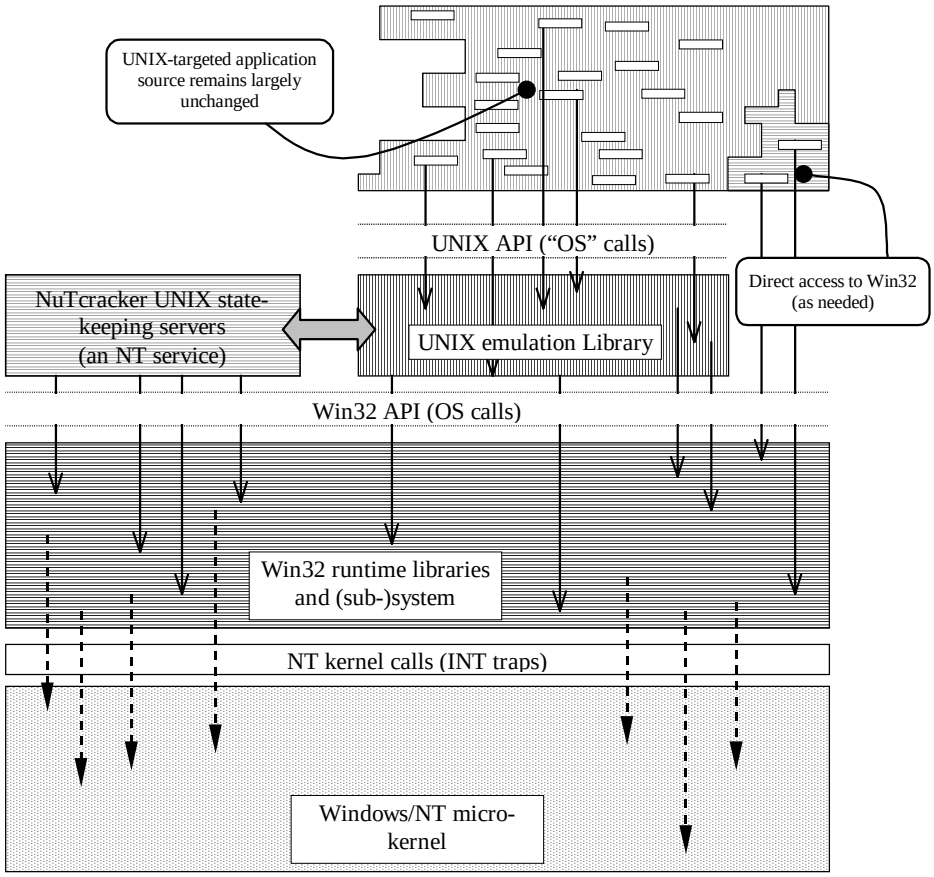


Figure 7: NuTcracker Architecture

The UNIX emulation library can be viewed as a pass-through conversion layer causing additional overhead. In practice, however, many of the UNIX emulation library features (like `sprintf()`, `memcpy()`, `sscanf()`, etc.) are just alternate, standards-compliant implementations of UNIX API calls. As such, their efficiency is about the same as that of a "real" UNIX system. For those specific calls that do reach below into the Win32 runtime or kernel (like `open()`, shared memory, sockets, etc.), there is some additional overhead.

Interix Architecture

The figure below shows a minimally-changed application employing the Interix framework. Note that the application cannot make Win32 API calls, but can invoke or reach Win32 applications, services or GUIs via traditional UNIX inter-process communication—sockets, RPC, shared memory, file systems entries, etc.

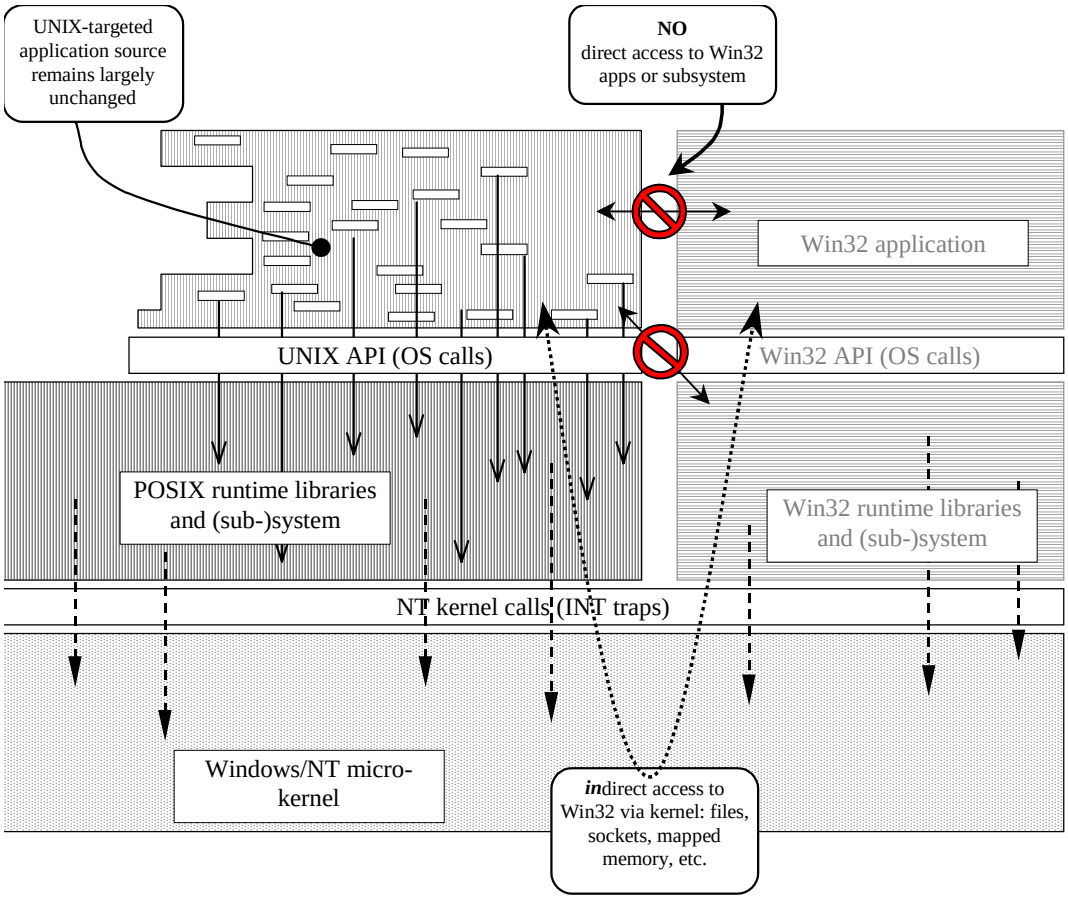


Figure 8: Interix Architecture

A POSIX runtime library is employed in place of the Win32 runtime libraries, with similar efficiencies in reaching the OS kernel when that is necessary. Because the Windows/NT kernel implements several features to enable or support POSIX, operations that are hard to do in the Win32 subsystem are not impeded. Examples include `fork()` and its associated process address space duplication, and POSIX-semantic file system protections.

(Reverse) Migration via Win32 API Emulation

A number of factors may suggest a very different strategy:

- The UNIX code or product line may be being end-of-life'd
- Win32 programmers and GUI experts may be easier to hire or train
- The Win32 GUI feature set is richer and may be more applicable or helpful to the application's users
- Market segment forecasts may show Windows products to become the majority sooner rather than later
- Recently-acquired products that are key to a product line may be Win32 code

(Note: we're *not* defining a new, unique API nor using some company's proprietary one; we're using a de-facto standard one: the MSFT Win32 API)

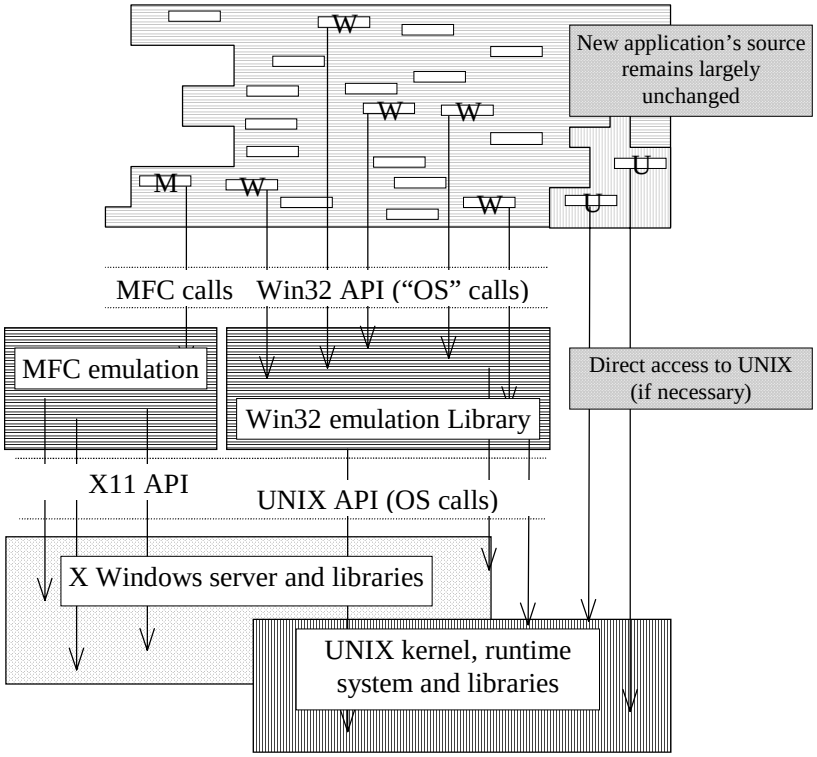
Any of these situations may argue for reverse migration: carrying Win32 code to UNIX boxes. There are at least three methods[10] for doing so:

- Win32-on-UNIX native binary code development environments: MainSoft's *MainWin*, Bristol's *Wind/U* and Willows' *Twin*
- PC processor emulation: Insignia's SoftWindows 95
- Actual Win32 running on a hardware co-processor

[10] SunSoft (Wabi) and others make Win16-only emulation and/or dynamic binary translation layers; they are not discussed here.

Win32-on-UNIX Binary Code Development

This method is the most efficient for running Win32 code on UNIX. All code is compiled to native binaries (but involves use of interface translation libraries). This is the best choice if your product line will be primarily targeted toward Windows, but need to occasionally be sold on UNIX.



Bibliography

Lowe, Andrew, *Porting UNIX Applications to Windows NT*, Macmillan Technical Publishing, 1997; ISBN 1-57879-004-3

NuTCRACKER Porting Guide and *Programmer's Reference Manual*, DataFocus, Inc., 1997

Interix "`man`" pages and other documentation, Softway Systems, Inc., 1998